PARASOFT®
We make software work.™

Top 10 User Mistakes with Static Analysis

Sate IV

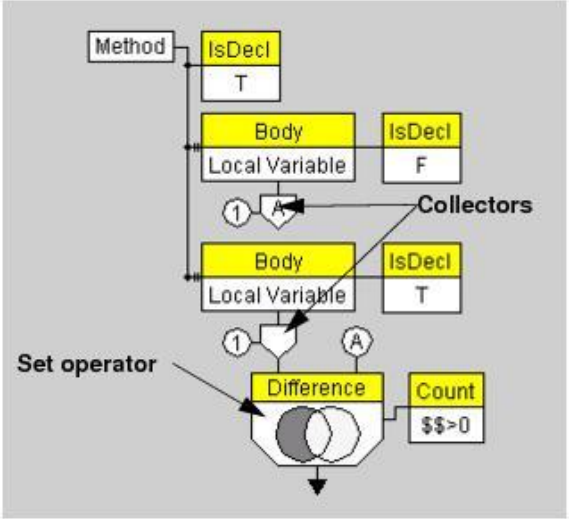
March 2012

About Parasoft

- Founded in 1987
- 27+ Patents for automated quality processes
- Build quality into the process
- Static Analysis tools since 1994

What IS Static Analysis?

- Variety of methods
  - Peer Review / Manual Code Review / Code Inspection
  - Pattern-based code scanners
  - Flow-based code scanners
  - Metrics-based code scanners
  - Compiler / build output



Number 10: Developers

10) Developers not included in process evolution

- Developer Insights
  - Rules / Issues drive need
  - Workflow
  - Usability
  - Correctness / Noise
- Will our engineers really adopt it and use it?
- Is this a long-term solution?

Code Analysis Perceptions

- "Static analysis is a pain"
- False positives has varying definitions
  - I don't like it
  - It was wrong

Pattern based false positives

- True false positives generally rule deficiency
- Context
  - Does this apply here and now?
  - In-code suppressions to document decision

Flow Analysis False Positives

- False positives are inevitable
- Finds real bugs
- Flow analysis is not comprehensive

Number 9: Expectations

PARASOFT

9) Wrong expectations

- Why do static analysis?
  - Because it's the right thing?
  - Increase quality?
  - Decrease costs?
  - Reduce development time?
- Flow analysis is enough
- When will it pay-off?
- How can I tell it's paying off?

Number 8: Approach

8) Taking an audit approach

- Running SA on all your code (Don't)
- It's all about the reports (Or is it?)

Number 7: Too Much

PARASOFT

7) Starting with too many rules

- Static Analysis is about process
- It's incremental
- Avoid biting off more than you can chew
- Avoid any rule you won't stop the build for

Don’t Get Run Over

- Same set of rules for everyone
- Small set of rules
- Less rules that are followed is better than more that are not
- If you wouldn’t fix it, don’t check for it

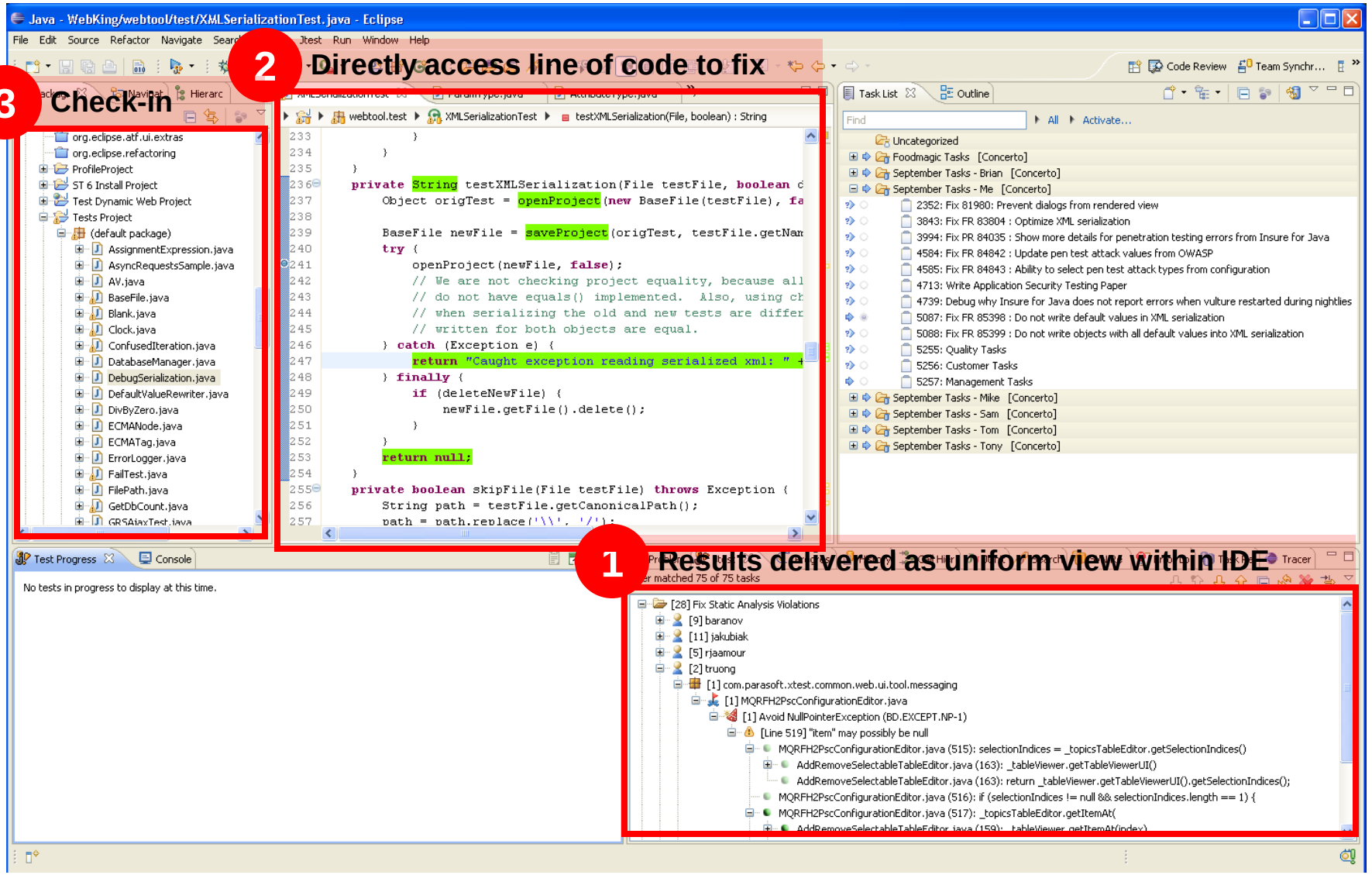
Number 6: Workflow

PARASOFT

6) Workflow integration

- Has to work with your development UI
- Same configuration for desktop and server
- Minimize negative impact
- Minimize time to find / fix violations

Results within IDE



Number 5: Training

5) Lack of sufficient training

- How to install the tool
- How to configure the tool
- How to setup the build
- How to run the tool
- How to mitigate violations
- How/when to suppress

Number 4: Process

4) No defined process

- Developers are not necessarily process experts
- Process should minimize impact of SA
- Consistent for teams and projects
- Vetted in a pilot project

Number 3: Automation

3) No automated process enforcement

- Reduce effort
- Consistency
- Compliance

Number 2: Policy

2) Lack of a clear policy

- What teams need to do SA?
- What projects require SA?
- What rules are required?
- What amount of compliance?
- When can you suppress?
- How to handle legacy code?
- Do you ship with SA violations?

Number 1: Management

1) Lack of management buy-in

- Requirements
- Allowed time
- Understanding of the ROI
- Enforcement

The Whole Top 10

- 10) Developers not included in process evolution
- 9) Wrong expectations
- 8) Taking an audit approach
- 7) Starting with too many rules
- 6) Workflow integration
- 5) Lack of sufficient training
- 4) No defined process
- 3) No automated process enforcement
- 2) Lack of a clear policy
- 1) Lack of management buy-in

Honorable Mention: The Wrong Stuff

PARASOFT

- Wrong Tool
- Wrong Process
  - Email reports
  - Blocking
  - Painful CI workflow
- Wrong Rules
  - Unimportant rules
  - Too many rules
- Wrong Code
  - Legacy strategy
  - Don’t test what you won’t / can’t change

Honorable Mention: What's Lacking

- Lack of management buy-in
  - The edict
  - Allowed time & budget
- Lack of development buy-in
  - Willful non-compliance
- Lack of training

Q&A / Further Reading

- *Automated Defect Prevention*
  (Huizinga & Kolawa)
  …Principles and processes to improve the software development process.
- *Effective C++ / More Effective C++*
  (Meyers)
  …Definitive work on proper C++ design and programming.
- *Effective Java*
  (Bloch)
  …Best-practice solutions for programming challenges.
- *Design Patterns*
  (Gamma, Helm, Johnson, Vlissides)
  …Timeless and elegant solutions to common problems.